RH188 EXAM PAPER

-

Question-1.

Running Simple Containers

Important: Unless explicitly asked for, do not customize the images used in this task. ACME Corporation is currently experimenting with the new microservices paradigm for their software, they need help containerizing their applications and are looking for a demonstration of the technology and particularly podman.
As an initial demonstration, you will show them nginx running as a containerized web server.

Tasks

Start a container with the following parameters:

1. Uses the image oci-registry:5000/nginx:latest
2. The container must be named acme-demo-html
3. The container must be detached from the command line
4. The container's port 80 must be mapped to external port 8001
5. The container must serve the contents of /home/desktop/workspace/acme-demo-html/index.html when accessed via http://desktop:8001

Hint – the container internally serves /usr/share/nginx/index.html at this URL
Note: http://desktop:8001 must always show the current contents of /home/desktop/workspace/acme-demo-html/index.html with no need to rebuild or restart the container.
Very important: The container acme-demo-html must persist after completing the task.

Test work by doing the following

1. With the container running browse to http://desktop:8001 and you should see in your browser the massage:

   “Welcome in Podman”

2. Having verified that you see the correct message, alter /home/desktop/workspace/acme- demo-html/index.html to something different (e.g., run “date > /home/desktop/workspace/acme-demo-html/index.html”) then refresh the browser and you should see your new contents.

Solution

1. podman pull oci-registry:5000/nginx:latest
2. podman image ls
3. podman run -dit --name=acme-demo-html -p 8001:80 -v /home/desktop/workspace/acme-demo-html:/usr/share/nginx/html: Z oci-registry:5000/nginx:latest
4. podman container ls
5. podman inspect acme-demo-html
6. curl http://localhost:8001

Question2.

Interacting with Running Containers

Important: Unless explicitly asked for, do not customize the Images used in this task.
ACME Corporation starts to be more interested in container technology, they would like to know more about it and particularly they ask how would it be possible to change the behaviour of live running containers.
To accomplish this task, you will update a configuration file provided that updates the location of the web

content, apply the new content and then interact the web server process inside the container to reload itself so that the new configuration applies and serves the new content.

Tasks

Start a container with the following parameters:

1. Use the image oci-registtry:5000/nginx:latest
2. The container must be named acme-demo-nginx
3. The container must be detached from the command line
4. The container's port 80 must be mapped to external port 8002
5. Copy the directory /home/desktop/workspace/acme-demo-nginx/html and its contents inside the container into /html
6. Copy the file /home/desktop/workspace/acme-demo-nginx/conf/nginx/default.conf over the container file at /etc/nginx/conf.d/default.conf
7. Execute the command nginx -s reload inside the running container

Very Important: The container acme-demo-nginx must persist after completing the task.

Test your work by doing the following

1. With the content running, browse to http://desktop:8002 and verify that you can see the default Nginx welcome page
2. Copy the file as indicated and run the command inside the container as instructed, then refresh the browser at http://desktop:8002

   You should see in your browser the new content with the message:

   "Welcome to ACME Corporation"

Solution

1. podman pull oci-registry:5000/nginix: latest
2. podman image ls
3. podman run -dit --name acme-demo-nginx -p 8002:80 oci-registtry:5000/nginx:latest
4. podman container ls
5. podman cp /home/desktop/workspace/acme-demo-nginx/html acme-demo-nginx:/html
6. podman cp /home/desktop/workspace/acme-demo-nginx/conf/nginx/default.conf acme-demo-nginx:/etc/nginx/conf.d/default.conf
7. podman exec –it acme-demo-nginx bash
8. nginx – s reload
9. podman inspect acme-demo-nginx

Question3.

Injecting Variables into Containers

Important: Unless explicitly asked for, do not customize the Images used in this task.
ACME Corporation development process includes passing dynamic runtime information when running their applications, they would like to know whether that is possible using podman as well as running different versions of the application running concurrently.
To accomplish this task, you will run different containers that show different content when based on environment variables.

Hint – the two containers are not expected to be run at the same time, making them work together is NOT part of this task.

Tasks

1. Create 2 containers with the image oci-registry:5000/nginx:acme
2. The container names must be:
   acme-demo-runtime_1
   acme-demo-runtime_2

3. Container acme-demo-runtime_1 is passed the following environment variable:
   Variable name: WELCOME_MASSAGE
   Variable value: ACME_Container_1
4. Container acme-demo-runtime_2 is passed the following environment variable:
   Variable name: WELCOME_MASSAGE
   Variable value: ACME_Container_2
5. Makes the port 8080 externally available and mapped to the container port 80 for all and each of the containers.
6. The containers must be detached from the command line.
7. The containers must coexist.

Test your work by doing following

1. Open a terminal and start the container acme-demo-runtime_1 browse to http://desktop:8080

 If the container is running properly, you should see in your browser the message:

"ACME_Container_1"

2. Now run the container acme-demo-runtime_2 and the browser should display:

"ACME_Container_2"

Solution

1. podman pull oci-registry:5000/nginx:acme
2. podman run -dit --name acme-demo-runtime_1 -p 8080:80 -e WELCOME_MESSAGE=ACME_Container _1 oci-registry:5000/ngnix:acme
3. podman container ls
4. podman inspect acme-demo-runtime_1
5. podman exec acme-demo-runtime_1 env
6. podman stop acme-demo-runtime_1
7. podman run -dit --name acme-demo-runtime_2 -p 8080:80 -e WELCOME_MESSAGE=ACME_Container _2   oci-registry:5000/ngnix:acme
8. podman exec acme-demo-runtime_2 env
9. podman inspect acme-demo-runtime_2
10. podman container ps -a

Question 4.

Build and Manage Container Images

Important: Unless explicitly asked for, do not customize the Images used in this task.
ACME Corporation has bought into the container technology. They would like to containerize their backend databases and migrate their development lifecycle using podman.
You are tasked to provide a demonstration on how it is possible to set up a containerized database that populates itself from data and also how it is possible to create simple backup using a containerized database backup tool.

Tasks

Update /home/desktop/workspace/acme-db/Containerfile.acme-db and add the necessary build instruction or instructions so that

1. The base image is mariadb:latest
2. Accepts the following build arguments
   DB_ROOT_PASSWORD
   DB_ACME_DATABASE

3. Pass the following environment variables:
    Variable name: MARIADB_ROOT_PASSWORD -> Variable value the value of DB_ROOT_PASSWORD
    Variable name: MARIADB_DATABASE-> Variable value the value of DB_ACME_DATABASE
4. Copies the file sql/acmeData.sql into the container directory /docker-entrypointinitdb.d
5. Build an image tagged acme-db:latest using the file /home/desktop/workspace/acme-db/Containerfile.acme-db that you have updated as build instructions and pass the following build arguments:
    Argument name: DB_ROOT_PASSWORD -> Argument value: acme
    Argument name: DB_ACME_DATABASE -> Argument value: acme
6. Push the image acme-db:latest to the private registry at acmeregistry:5000/acmedb:latest

Update /home/desktop/workspace/acme-db/Containerfile.acme-db-exporter and add the necessary build instruction or instructions so that:

1. The base image is maridb:latest
2. Copies the script scripts/exporter.sh into /scripts
3. Defines the working directory as /scripts
4. The container starts by executing the script /scripts/exporter.sh

5. Build another image tagged acme-db-exporter:latest using the file Containerfile.acme-db-exporter as build instructions.
6. Push the image acme-db-exporter:latest to the private registry at acmeregistry:5000/acme-db-exporter:latest

Very Important: The image build and pushed to the private registry must persist upon completion of this task.

Solution

First image
1. cd /home/desktop/workspace/acme-db/
2. vi Containerfile.acme-db

*Inside the Containerfile.acme-db file

```
FROM  mariadb:latest
ARG DB_ROOT_PASSWORD
ARG DB_ACME_DATABASE
ENV  MARIADB_ROOT_PASSWORD=$
{DB_ROOT_PASSWORD}
ENV  MARIADB_DATABASE=${DB_ACME_DATABASE}
COPY ./sql/acmeData.sql  /docker-entrypoint-intdb.d/
```

3. podman build --build-arg DB_ROOT_PASSWORD =acme --build-arg
DB_ACME_DATABASE=acme --file
Containerfile.acme.db -t acme-db: latest
4. podman image inspect acme-db:latest
5. podman tag acme-db:latest acmeregistry:5000/ acme-db:latest
6. podman push acmeregistry:5000/acme-db:latest

Second image-->

1. vi Containerfile.acme-db-exporter
   *Inside the Containerfile.acme-db file

```
FROM mariadb:latest
COPY ./scripts/exporter.sh  /scripts/
WORKDIR  /scripts
CMD /scripts/exporter.sh
```

2. podman build -f Containerfile.acme-db-exporter -t acme-db-exporter:latest .
3. podman tag acme-db-exporter:latestt acmeregistry:5000/ acme-db-exporter:latest
4. podman push acmeregistry:5000/ acme-db-exporter:latest
5. podman image inspect acme-db-exporter:latest

Test your work by doing the following

1. First image
--> podman run –dit –name test-acme-db –p 3306:3306 acme-db:latest
--> podman container ls
2. Second image
--> podman run –dit –name test-acme-db-exporter -v /home/workspace/acme-db/export:/export acme-db-exporter:latest
--> podman container ls

Question 5.

Run and Manage Multi-Container Applications

Important: Unless explicitly asked for do not customize the images used in this task.
ACME Corporation is very happy with how podman can make their development lifecycle easier. They are currently working on a small 3-tier stack and they would like you to give training to their developers on containerize the stack.
Your goal is to provide them a ready application stuck with a backend, an application and a frontend web server that will handle the application requests.

Tasks

1. Create a network named acme-wp
2. Create the volume for the database, name it acme-wp-backend
3. Create another volume for the Wordpress application, name it acme-wp-app
4. Start the Database container with the following setting:
   (i) Uses the provided image oci-registry:5000/acme:wp-backend
   (ii) Runs detached from the command line.
   (iii) Must be named acme-wp-backend
   (iv) Must attach the volume acme-wp-backend to the container at the directory /var/lib/mysql
   (v) Must attach to the network acme-wp
5. Start the Wordpress application container with the following setting:
   (i) Uses the provided image oci-registry:5000/acme:wp-app
   (ii) Runs detached from the command line.
   (iii) Must be named acme-wp-app
   (iv) Must attach the volume acme-wp-app to the container at the directory /var/www/html
   (v) Must attach to the network acme-wp
6. Start the web server container with the following setting:
   (i) Uses the provided image oci-registry:5000/acme:wp-frontend
   (ii) Runs detached from the command line.

(iii) Must be named acme-wp-frontend
(iv) Must attach the volume acme-wp-app to the container at the directory /var/www/html
(v) Must attach to the network acme-wp
(vi) Makes the port 8003 externally available and mapped to the container port 80

Important: The containers acme-wp-backend, acme-wp-app, acme-wp-frontend, the volumes acme-wp-backend, acme-wp-app and the network acme-wp must all persist after completing the task.

Test work by doing the following

1. Start all the application containers as indicated.
2. When they are all running and healthy, browse to http://desktop:8003 and verify that you can see a working Wordpress site Welcome to ACME Corporation and also a post titled Hello world!

Solution

1. podman network create acme-app
   podman network ls
   podman network inspect acme-app
2. podman volume create acme-wp-backend
3. podman volume create acme-wp-app
   podman volume ls
   podman volume inspect acme-wp-backend
   podman volume inspect acme-wp-app

4. podman run -dit --name acme-wp-backend -v acme-wp-backend:/var/lib/mysql --net acme-wp oci-registry:5000/acme:wp-backend
   podman container ls
   podman inspect acme-wp-backend
5. podman run -dit --name acme-wp-app -v acme-wp-app:/var/www/html   --net acme-wp oci-registry:5000/acme:wp
   podman container ls
   podman inspect acme-wp-app
6. podman run -dit --name acme-wp-frontend -v acme-wp-app:/var/www/html -p 8003:80   --net acme-wp oci-registry:5000/acme:wp-frontend
   podman container ls
   podman inspect acme-wp-frontend
7. podman ps -a

Question6.

Troubleshot Containers

ACME Corporation has developed a new version of their application stack, however they are running into some problems when they try to bring up to all the containers. Using the provided images, you will spin up a working container stack as you troubleshoot and fix the problems.

Tasks

1. Create a network named acme-troubles
2. Create the volume for the database, name it acme-wp-backend-ts
3. Create another volume for the Wordpress application, name it acme-wp-app-ts
4. The database backend container must follow at minimum the following configuration:
   (i) Must use the provided image oci-registry:5000/acme:wp-backend-broken
   (ii) Must be named acme-wp-backend-ts
   (iii) Must use the network acme-troubles
   (iv) Must attach the volume acme-wp-backend-ts accessible on the container at /var/lib/mysql
5. The Wordpress application container must follow at minimum this parameters:
   (i) Must use the provided image oci-registry:5000/acme:wp-app-broken
   (ii) Must be named acme-wp-app-ts
   (iii) Must use the network acme-troubles
   (iv) Must attach the volume acme-wp-app-ts accessible on the container at /var/www/html
6. The Webserver application container must follow at minimum this parameters:
   (i) Must use the provided image oci-registry:5000/acme:wp-frontend-broken
   (ii) Must be named acme-wp-frontend-ts
   (iii) Must use the network acme-troubles

(iv)Must attach the volume acme-wp-app-ts accessible on the container at /var/www/html

(v)Makes the port 8004 externally available and mapped to the container port 80

Hints:

1. All the issues are fully resolvable running the containers and using the right command line parameters to fix the problems and you need only podman to investigate and troubleshoot.
2. The Wordpress application container can read the environment variable WORDPRESS_DEBUG=true to provide extended debugging information through the web browser.
3. Database image Containerfile for reference
4. Wordpress image Containerfile for reference
5. Web server image Containerfile for reference

Important: All the containers, volumes and network must persist after completion of the task.

Test work by doing the following

Identity the problem or problems with each container. Apply the solution to run the containers properly using the command line. Ones stack is up, browse to http://desktop:8094 and verify that you can see the Wordpress site with the title ACME IS FIXED

Solution

This question is same as above but it will show error in the browser so check logs.

```
-->podman logs acme-wp-app-ts
```

In the logs it will show something like "fixme".

Find the error and solve it.